Click to verify

Python3 实例 定义一个列表，并将它翻转。 例如： 翻转前：list = [10, 11, 12, 13, 14, 15] 翻转后：[15, 14, 13, 12, 11, 10] def Reverse(lst):   return [ele for ele in reversed(lst)] lst = [10, 11, 12, 13, 14, 15] print(Reverse(lst)) 以上实例输出结果为： [15, 14, 13, 12, 11, 10] def Reverse(lst):   lst.reverse()   return lst lst = [10, 11, 12, 13, 14, 15] print(Reverse(lst)) 以上实例输出结果为： [15, 14, 13, 12, 11, 10] def Reverse(lst):   new_lst = lst[::-1]   return new_lst lst = [10, 11, 12, 13, 14, 15] print(Reverse(lst)) 以上实例输出结果为： [15, 14, 13, 12, 11, 10] Python3 实例 nvm (Node Version Manager) 是一个非常有用的工具，可以让您在同一台机器上安装和管理多个 Node.js 版本。 为什么需要 nvm？ 不同项目可能需要不同版本的 Node.js 测试应用在不同 Node.js 版本下的兼容性 方便升级和降级 Node.js 版本 安装 nvm 在 macOS/Linux 上安装 nvm： # 使用 curl 安装 curl -o- | bash # 或使用 wget 安装 wget -qO- | bash # 重新加载 shell 配置 source ~/.bashrc # 或 source ~/.zshrc 在 Windows 上安装 nvm-windows： nvm 常用命令： # 查看 nvm 版本 nvm --version # 列出所有可安装的 Node.js 版本 nvm list-remote # Windows 上使用 nvm list available # 安装最新的 LTS 版本 nvm install --lts # 安装特定版本 nvm install 18.17.0 nvm install 16.20.1 # 列出已安装的版本 nvm list # 或 nvm ls # 切换到特定版本 nvm use 18.17.0 # 设置默认版本 nvm alias default 18.17.0 # 查看当前使用的版本 nvm current # 卸载特定版本 nvm uninstall 16.20.1 实际使用示例： # 场景：为不同项目使用不同 Node.js 版本 # 项目 A 使用 Node.js 18 cd project-a nvm use 18.17.0 node --version # v18.17.0 # 项目 B 使用 Node.js 16 cd ../project-b nvm use 16.20.1 node --version # v16.20.1 # 为项目指定 Node.js 版本 echo "18.17.0" > .nvmrc nvm use # 自动使用 .nvmrc 中指定的版本 验证安装是否成功 创建第一个 Node.js 程序： 创建一个名为 hello.js 的文件： // hello.js console.log('Hello, Node.js!'); console.log('Node.js 版本:', process.version); console.log('当前工作目录:', process.cwd()); console.log('操作系统:', process.platform); 预期输出： Hello, Node.js! Node.js 版本: v18.17.0 当前工作目录: /Users/username/projects 操作系统: darwin 检查全局安装路径： # 查看 npm 全局包安装路径 npm config get prefix # 查看 npm 配置 npm config list # 查看 Node.js 安装路径 which node # Windows 上使用 where node Linux 命令大全 Linux ls (英文全拼： list directory contents) 命令用于显示指定工作目录下之内容 (列出目前工作目录所含的文件及子目录)。 语法 ls [-alrtAFR] [name...] 参数：参数说明-a 或 --all显示所有文件 (包括以 . 开头的隐藏文件)。-A 或 --almost-all显示除 . 和 .. 外的所有文件 (包括隐藏文件)。-l以长格式 (详细信息) 列出文件 (权限、所有者、大小、修改时间等)。-h 或 --human-readable与 -l 一起使用时，以人类可读的格式显示文件大小 (如 KB、MB)。-t按修改时间排序 (最新优先)。-r 或 --reverse反向排序 (配合 -t、-S 等使用)。-S按文件大小排序 (大文件优先)。-R 或 --recursive递归列出子目录内容。-F 或 --classify在文件名后附加标识符 (如 / 表示目录 , * 表示可执行文件)。--color彩色输出 (通常默认启用 , --color=auto)。-i 或 --inode显示文件的 inode 编号。-n 或 --numeric-uid-gid以数字形式显示 UID 和 GID (替代用户名和组名)。-d 或 --directory仅显示目录本身，而非其内容 (常用于配合 -l)。-1每行只显示一个文件 (默认在终端宽度不足时自动启用)。-m以逗号分隔的列表形式显示文件。-Q 或 --quote-name用引号括住文件名 (适用于含空格的文件名)。--group-directories-first先显示目录，后显示文件。--time-style=自定义时间显示格式 (如 +%Y-%m-%d)。 ls -l # 以长格式显示当前目录中的文件和目录 ls -a # 显示当前目录中的所有文件和目录，包括隐藏文件 ls -lh # 以人类可读的方式显示当前目录中的文件和目录大小 ls -t # 按照修改时间排序显示当前目录中的文件和目录 ls -R # 递归显示当前目录中的所有文件和子目录 ls -l /etc/passwd # 显示/etc/passwd文件的详细信息 实例 详细列出当前目录所有文件 (含隐藏文件)： ls -la 按大小反向排序文件 (大文件优先)： ls -lShr 递归列出 /var/log 目录内容，并显示人类可读的文件大小： ls -lhR /var/log 仅显示目录的详细信息 (不递归)： ls -ld /etc 按修改时间排序 (最新文件最后显示)： ls -ltr 列出根目录下的所有目录： # ls / bin dev lib media net root srv upload www boot etc lib64 misc opt sbin sys usr home lost+found mnt proc selinux tmp var 将 /bin 目录以下所有目录及文件详细资料列出: ls -lR /bin 当文件名包含空格、特殊字符或者开始字符为破折号时，可以使用反斜杠 (\ ) 进行转义，或者使用引号将文件名括起来。例如： ls "my file.txt" # 列出文件名为"my file.txt"的文件 ls my\ file.txt # 列出文件名为"my file.txt"的文件 ls -- -filename # 列出文件名为"-filename"的文件 ls 命令还可以使用通配符进行模式匹配，例如 * 表示匹配任意字符，? 表示匹配一个字符，[...] 表示匹配指定范围内的字符。例如： ls *.txt # 列出所有扩展名为.txt的文件 ls file?.txt # 列出文件名为file?.txt的文件，其中?表示任意一个字符 ls [abc]*.txt # 列出以a、b或c开头、扩展名为.txt的文件 列出目前工作目录下所有名称是 s 开头的文件，越新的排越后面: ls -ltr s* 在使用 ls -l 命令时，第一列的字符表示文件或目录的类型和权限。其中第一个字符表示文件类型，例如： - 表示普通文件 d 表示目录 l 表示符号链接 c 表示字符设备文件 b 表示块设备文件 s 表示套接字文件 p 表示管道文件 在使用 ls -l 命令时，第一列的其余 9 个字符表示文件或目录的访问权限，分别对应三个字符一组的 rwx 权限。例如： r 表示读取权限 w 表示写入权限 x 表示执行权限 - 表示没有对应权限 前三个字符表示所属组的权限，后三个字符表示其他用户的权限。例如： -rw-r--r-- 1 user group 4096 Feb 21 12:00 file.txt 表示文件名为file.txt的文件，所有者具有读写权限，所属组和其他用户只具有读取权限。 查找最近修改的文件： ls -lt | head -5 显示最近修改的 5 个文件。 统计文件数量： ls | wc -l 统计当前目录下的文件数量(不包括隐藏文件)。 注意事项 ls 命令的输出颜色可以通过 --color 选项控制： 蓝色：目录 绿色：可执行文件 红色：压缩文件 青色：链接文件 黄色：设备文件 在脚本中使用 ls 时要注意，直接解析 ls 的输出可能不可靠，建议使用其他方法。 不同 Linux 发行版的 ls 命令可能有细微差别，可以通过 man ls 查看具体帮助文档。 Linux 命令大全 Java 集合框架 ArrayList 类是一个可以动态修改的数组，与普通数组的区别就是它是没有固定大小的限制，我们可以添加或删除元素。 ArrayList 继承了 AbstractList ，并实现了 List 接口。 ArrayList 类位于 java.util 包中，使用前需要引入它，语法格式如下： import java.util.ArrayList; // 引入 ArrayList 类 ArrayList objectName =new ArrayList(); // 初始化 E: 泛型数据类型，用于设置 objectName 的数据类型，只能为引用数据类型。 objectName: 对象名。 ArrayList 是一个数组队列，提供了相关的添加、删除、修改、遍历等功能。 添加元素 ArrayList 类提供了很多有用的方法，添加元素到 ArrayList 可以使用 add() 方法: import java.util.ArrayList; public class RunoobTest {   public static void main(String[] args) {   ArrayList sites = new ArrayList();   sites.add("Google");   sites.add("Runoob");   sites.add("Taobao");   sites.add("Weibo");   System.out.println(sites);   } } 以上实例，执行输出结果为： [Google, Runoob, Taobao, Weibo] 访问元素 访问 ArrayList 中的元素可以使用 get() 方法： import java.util.ArrayList; public class RunoobTest {   public static void main(String[] args) {   ArrayList sites = new ArrayList();   sites.add("Google");   sites.add("Runoob");   sites.add("Taobao");   sites.add("Weibo");   System.out.println(sites.get(1)); // 访问第二个元素   } } 注意：数组的索引值从 0 开始。 以上实例，执行输出结果为： Runoob 修改元素 如果要修改 ArrayList 中的元素可以使用 set() 方法， set(int index, E element) 方法的第一个参数是索引 (index) ，表示要替换的元素的位置，第二个参数是新元素 (element) ，表示要设置的新值： import java.util.ArrayList; public class RunoobTest {   public static void main(String[] args) {   ArrayList sites = new ArrayList();   sites.add("Google");   sites.add("Runoob");   sites.add("Taobao");   sites.add("Weibo");   sites.set(2, "Wiki"); // 第一个参数为索引位置，第二个为要修改的值   System.out.println(sites);   } } 以上实例，执行输出结果为： [Google, Runoob, Wiki, Weibo] 删除元素 如果要删除 ArrayList 中的元素可以使用 remove() 方法： import java.util.ArrayList; public class RunoobTest {   public static void main(String[] args) {   ArrayList sites = new ArrayList();   sites.add("Google");   sites.add("Runoob");   sites.add("Taobao");   sites.add("Weibo");   sites.remove(3); // 删除第四个元素   System.out.println(sites);   } } 以上实例，执行输出结果为： [Google, Runoob, Taobao] 计算大小 如果要计算 ArrayList 中的元素数量可以使用 size() 方法： import java.util.ArrayList; public class RunoobTest {   public static void main(String[] args) {   ArrayList sites = new ArrayList();   sites.add("Google");   sites.add("Runoob");   sites.add("Taobao");   sites.add("Weibo");   System.out.println(sites.size());   } } 以上实例，执行输出结果为： 4 迭代数组列表 我们可以使用 for 来迭代数组列表中的元素： import java.util.ArrayList; public class RunoobTest {   public static void main(String[] args) {   ArrayList sites = new ArrayList();   sites.add("Google");   sites.add("Runoob");   sites.add("Taobao");   sites.add("Weibo");   for (int i = 0; i < sites.size(); i++) {   System.out.println(sites.get(i));   }   } } 以上实例，执行输出结果为： Google Runoob Taobao Weibo 也可以使用 for-each 来迭代元素： import java.util.ArrayList; public class RunoobTest {   public static void main(String[] args) {   ArrayList sites = new ArrayList();   sites.add("Google");   sites.add("Runoob");   sites.add("Taobao");   sites.add("Weibo");   for (String i : sites) {   System.out.println(i);   }   } } 以上实例，执行输出结果为： Google Runoob Taobao Weibo 其他的引用类型 ArrayList 中的元素实际上是对象，在以上实例中，数组列表元素都是字符串 String 类型。 如果我们要存储其他类型，而 只能为引用数据类型，这时我们就需要使用到基本类型的包装类。 基本类型对应的包装类表如下： 基本类型引用类型 booleanBoolean byteByte shortShort intInteger longLong floatFloat doubleDouble charCharacter 此外，BigInteger、BigDecimal 用于高精度的运算，BigInteger 支持任意精度的整数，也是引用类型，但它们没有相对应的基本类型。 ArrayList li=new ArrayList(); // 存放整数元素 ArrayList li=new ArrayList(); // 存放字符元素 以下实例使用 ArrayList 存储数字(使用 Integer 类型): import java.util.ArrayList; public class RunoobTest {   public static void main(String[] args) {   ArrayList myNumbers = new ArrayList();   myNumbers.add(10);   myNumbers.add(15);   myNumbers.add(20);   myNumbers.add(25);   for (int i : myNumbers) {   System.out.println(i);   }   } } 以上实例，执行输出结果为： 10 15 20 25 ArrayList 排序 Collections 类也是一个非常有用的类，位于 java.util 包中，提供的 sort() 方法可以对字符或数字列表进行排序。 以下实例对字母进行排序： import java.util.ArrayList; import java.util.Collections; // 引入 Collections 类 public class RunoobTest {   public static void main(String[] args) {   ArrayList sites = new ArrayList();   sites.add("Taobao");   sites.add("Wiki");   sites.add("Runoob");   sites.add("Weibo");   sites.add("Google");   Collections.sort(sites); // 字母排序   for (String i : sites) {   System.out.println(i);   }   } } 以上实例，执行输出结果为： Google Runoob Taobao Weibo Wiki 以下实例对数字进行排序： import java.util.ArrayList; import java.util.Collections; // 引入 Collections 类 public class RunoobTest {   public static void main(String[] args) {   ArrayList myNumbers = new ArrayList();   myNumbers.add(33);   myNumbers.add(15);   myNumbers.add(20);   myNumbers.add(34);   myNumbers.add(8);   myNumbers.add(12);   Collections.sort(myNumbers); // 数字排序   for (int i : myNumbers) {   System.out.println(i);   }   } } 以上实例，执行输出结果为： 8 12 15 20 33 34 Java ArrayList 方法 Java ArrayList 常用方法列表如下： 更多 API 方法可以查看： Java 集合框架 C++ 标准库提供了丰富的功能，其中 是一个非常重要的容器类，用于存储元素集合，支持双向迭代器。 是 C++ 标准模板库 (STL) 中的一个序列容器，它允许在容器的任意位置快速插入和删除元素。与数组或向量 ( ) 不同， 不需要在创建时指定大小，并且可以在任何位置添加或删除元素，而不需要重新分配内存。 语法 以下是 容器的一些基本操作： 包含头文件：#include 声明列表：std::list mylist; ，其中 T 是存储在列表中的元素类型。 插入元素：mylist.push_back(value); 删除元素：mylist.pop_back(); 或 mylist.erase(iterator); 访问元素：mylist.front(); 和 mylist.back(); 遍历列表：使用迭代器 for (auto it = mylist.begin(); it != mylist.end(); ++it) 特点 双向迭代： 提供了双向迭代器，可以向前和向后遍历元素。 动态大小：与数组不同， 的大小可以动态变化，不需要预先分配固定大小的内存。 快速插入和删除：可以在列表的任何位置快速插入或删除元素，而不需要像向量那样移动大量元素。 声明与初始化 的声明和初始化与其他容器类似： #include #include int main() { std::list lst1; // 空的list std::list lst2(5); // 包含5个默认初始化元素的list std::list lst3(5, 10); // 包含5个元素，每个元素为10 std::list lst4 = {1, 2, 3, 4}; // 使用初始化列表 return 0; } 实例 下面是一个使用 的简单示例，包括创建列表、添加元素、遍历列表和输出结果。 #include #include int main() {   // 创建一个整数类型的列表   std::list numbers;   // 向列表中添加元素   numbers.push_back(10);   numbers.push_back(20);   numbers.push_back(30);   // 访问并打印列表的第一个元素   std::cout